

Network Courier V2.1

Mail System Database Description

By Shannon Chan

Microsoft Confidential

Do Not Distribute Outside Of WorkGroup Applications

Contents

Contents.....	2
Foreword.....	4
1. Background.....	5
2. Concepts.....	6
3. Subdirectories.....	7
3.1. The ATT subdirectory.....	7
3.2. The FOLDERS subdirectory.....	8
3.2.1.....	Folders index f
3.2.2.....	Folder data fil
3.3. The GLB subdirectory.....	11
3.3.1. Directory of M:\COURIER2\GLB.....	11
3.3.2.....	ACCESS.GLB
3.3.3.....	ACCESS2.GLB
3.3.4.....	ACCESS3.GLB
3.3.5.....	CONTROL.GLB
3.3.6.....	GLOBAL.GLB
3.3.7.....	GROUP.GLB
3.3.8.....	GRPMEM.GLB
3.3.9.....	MASTER.GLB
3.3.10.....	NETPO.GLB
3.3.11.....	NETWORK.GLB
3.3.12.....	TID.GLB 23
3.3.13.....	Gateway address
3.4. The GRP subdirectory.....	23
3.5. The HLP subdirectory.....	24
3.6. The INF subdirectory.....	24
3.7. The KEY subdirectory.....	24
3.8. The LOG subdirectory.....	25
3.9. The MAI subdirectory.....	25
3.10. The MBG subdirectory.....	26
3.11. The MEM subdirectory.....	27
3.12. The NME subdirectory.....	27
3.12.1.....	Alias Record
3.13. The P1 subdirectory.....	28
3.14. The TPL subdirectory.....	29
3.14.1.....	Prompt 29
3.14.2.....	Start 29
3.14.3.....	Width 29
3.14.4.....	Edit 30
3.14.5.....	Internal 30
3.15. The USR subdirectory.....	30
3.16. The XTN subdirectory.....	30
NLS String Support.....	33
Outline.....	33
NLSCASE.C.....	33
Entry Points.....	34

Compiler Switches.....	34
NLSCOMP.C.....	34
Entry Points.....	35
Compiler Switches.....	35
NLSIS.C.....	36
Entry Points.....	36
Compiler Switches.....	36
NLSTRANS.C.....	36
Entry Points.....	37
Compiler Switches.....	37
NLSLOAD.C.....	38
Entry Points.....	38
Compiler Switches.....	38
External Comparison Tables.....	38
Comparison File Format.....	38
Comparison File Creation.....	39
 4. FIPS Message Format.....	40
4.1. Identifier.....	40
4.2. Length.....	40
4.3. Data Elements.....	41
4.3.1.....	Primitive Data
4.3.2.....	Constructor Dat
4.4. Message Fields.....	43
4.4.1.....	ATTACHMENTS opc
4.4.2.....	CC opcode
4.4.3.....	COPIES opcode
4.4.4.....	DATE opcode
4.4.5.....	FOLDATTACH opco
4.4.6.....	FROM Opcode
4.4.7.....	HOPCOUNT opcode
4.4.8.....	HOPTRACE opcode
MESSAGE FORMAT opcode.....	46
4.4.9.....	NLSTAG 46
4.4.10.....	PRIORITY opcode
4.4.11.....	SUBJECT opcode
4.4.12.....	TEXT opcode
4.4.13.....	TEXTATTR opcode
4.4.14.....	TEXTBORDER opco
4.4.15.....	TO opcode
 5. Algorithms.....	50
5.1. Logging in.....	50
5.2. Reading Mailbag Headers.....	50
5.3. Sending Mail To Mailbags.....	50
5.4. Checking For Outgoing Mail (External and gateways).....	51
5.5. Mailer.....	51
 Appendix A - Extending the Network Courier Database.....	52

Foreword

Parts of this document are based on the "Network Courier Mail System Database Description" for The Network Courier V1.0 written by Patrick Black. The original version of this document was lost some time ago. Some of the contents of that document are no longer applicable and so this document serves as a replacement for it.

1. Background

The Network Courier Version 2.0 was released in the spring of 1989. This version incorporated several major changes to the database structure as well as several new features in the individual programs which access the database. In the summer of 1990, Version 2.1 was released, incorporating some additional database changes as well as several new features. Since the purpose of this document is to describe the database, the focus will be on the architecture of the database, the structures used, and the design philosophy. Since there is no companion document for Version 2.0, the differences between 2.0 and 2.1 will be noted where appropriate.

The Network Courier is sold in units called "postoffices" which are essentially databases which provide electronic mail access for a limited number of users. Consumers Software currently markets four versions of The Network Courier. These versions are:

- 1) 6-User Starter kit
- 2) Single Postoffice
- 3) Internet
- 4) 500-User

Of these, the latter two allow transfer of mail between postoffices as well as between users on a single postoffice. Internetwork capability is also required in order for mail to be sent to other mail systems such as MCI Mail, IBM PROFS, X.400, etc.

The difference in database features between Internet and non-Internet versions are quite extensive. The focus here will be on Internet versions since a) they are more popular, and b) understanding the database of an Internet version implies knowledge of the database of a non-Internet version.

2. Concepts

Mail items are stored on the database in single files. Generally there are two ways of storing messages on a mail system database. One way is to maintain a separate copy of the mail for each person to whom the message is addressed. The other way is to maintain a single copy of the message with references or "links" to the file for each user in the TO or CC field. The Network Courier stores messages in the latter format which makes better use of a file server's disk storage. Mail items can be stored using server storage or by using local storage.

The Network Courier mail file format generally follows that described in FIPS publication 98, a variation of the CCITT X.409 message presentation format. These files are fully encrypted and structured such that new fields can be defined without sacrificing backward compatibility.

The references to the mail items are kept in files called "mailbags". A mailbag has a list of records in it, each representing a piece of mail. Included in the record is information such as the FROM, SUBJECT, PRIORITY, and DATE fields, as well as the name of the file containing the mail. Each user has his own mailbag and mail is delivered by putting references to the mail in the recipients' mailbags. If the mail is delivered to a user on another postoffice or gateway, the reference is put into a mailbag representing that postoffice or gateway and an external agent is responsible for delivering the mail to the external postoffice or gateway.

Mail can be stored into special files called folders. In this case, mail items are no longer single files and there is never more than one reference to a single folder item. If a piece of mail is stored twice into a folder then there are two physical copies of that mail item stored.

User accounts are maintained as records in a single file called ACCESS.GLB. There are parallel files called ACCESS2.GLB and ACCESS3.GLB which hold additional information for each user.

Information about other postoffices is maintained in files which have the ".XTN" extension name. A single XTN file contains information for every postoffice on a particular network. There is a single XTN file for every network defined. The files which store information about gateways and users on gateway systems (ie other e-mail systems) vary depending on the addressing and connection method. Generally, gateways which have an addressing scheme which maps easily to The Network Courier's network\postoffice\user scheme are stored similar to other Network Courier postoffices.

All data structures assume Intel byte-ordering with least significant byte first. Folders items are sorted using a case-insensitive, diacritic-insensitive comparison. Address lists are sorted using a case insensitive, diacritic sensitive comparison. All character strings stored in the database are generally assumed to be from the IBM code page 850 character set.

3. Subdirectories

On a newly installed postoffice, the data files are stored in a system of subdirectories. The current list of these subdirectories is:

ATT	- attachments
FOLDERS	- folders
GLB	- global files, error message files
GRP	- group definitions
HLP	- help files
INF	- information files
KEY	- key files (for mailbags)
LOG	- log files
MAI	- mail items
MBG	- mailbags
MEM	- group definitions
NME	- address lists
P1	- temporary files
TPL	- template files
USR	- external user lists
XTN	- postoffice definitions

Some gateways require additional directories which are created at installation time. The purpose of these directories is not covered in this document.

Generally the name of the subdirectory matches the filename extension of the files stored in that directory. Files in the XTN subdirectory all have names of the form nnnnnnnn.XTN, etc.

Most of the structure definitions for Network Courier database files can be found in the files ext.h and fm_def.h.

3.1. The ATT subdirectory

The ATT directory contains 16 subdirectories. These are named as follows and contain the following types of filenames:

AT0	- nnnnnnnn0.ATT
AT1	- nnnnnnnn1.ATT
AT2	- nnnnnnnn2.ATT
AT3	- nnnnnnnn3.ATT
AT4	- nnnnnnnn4.ATT
AT5	- nnnnnnnn5.ATT
AT6	- nnnnnnnn6.ATT
AT7	- nnnnnnnn7.ATT
AT8	- nnnnnnnn8.ATT
AT9	- nnnnnnnn9.ATT
ATA	- nnnnnnnnA.ATT
ATB	- nnnnnnnnB.ATT
ATC	- nnnnnnnnC.ATT
ATD	- nnnnnnnnD.ATT
ATE	- nnnnnnnnE.ATT

ATF - nnnnnnnF.ATT

Files which have the ".ATT" extension are attachment files that have been attached to a particular mail item. The reference to an attachment file is in the mail file itself. A single mail file can reference several attachments but a single attachment file cannot be referenced more than once. Attachment files are fully encrypted.

3.2. The FOLDERS subdirectory

This directory contains a single file called SERIALNO.IDX as well as two subdirectories called LOC and PUB. The subdirectory FOLDERS\LOC contains users' private folders and FOLDERS\PUB contains shared folders. The file SERIALNO.IDX contains a single long integer value which is used for generating unique folder file names.

FOLDERS\PUB contains two types of files:

FOLDROOT.IDX	- index of all shared folders
nnnnnnnn.FLD	- folder containing one or more messages

FOLDERS\LOC contains separate directories for each user. The directory name matches the mailbag name for any particular user. For instance, if a user's mailbag file is called 12345678.MBG then his folders will be in the directory FOLDERS\LOC\12345678. Each of these directories has essentially the same format as FOLDERS\PUB, except that instead of there being a FOLDROOT.IDX file, there is a file with the name nnnnnnnn.IDX where nnnnnnn is the filename prefix of the user's MBG file.

If a user's folders are on private storage, then his entire folder directory is moved to his local storage. His server folder directory still exists but is empty.

3.2.1. Folders index file format

Folders index (.IDX) files contain a single folder index header followed by any number of folder index records. The folders index file is essentially what you see when you bring up your list of folders in the Mail program.

3.2.1.1. Folders index header

offset	size	name	description
0	4	unsigned long uFirst	offset of the first record or 0
4	4	unsigned long uLast	offset of the last record or 0
8	4	long uSizeRecs	Total size of all records.
12	4	long uSizeDels	Total size of all deleted records.
16	2	unsigned uNoRecs	Number of records.
18	2	unsigned uDelRecs	Number of deleted records.
20	80	char SLACK[80]	

3.2.1.2. Folders index records

A series of folder index records follows the folder index header. Each record describes a folder.

offset	size	name	description
0	4	unsigned long uNext	next record
4	4	unsigned long uPrev	previous record
8	2	unsigned uDepth	0=deleted 1-nn= level
10	4	unsigned long ulWhere	which .fld file
14	2	unsigned bAttr	attributes on this folder
16	25	char szName[25]	used to store folder name
41	26	char szComment[26]	used to store folder comment
67	11	char szOwner[11]	Identifies the owner of the folder.
78	4	unsigned long ulNumMess	number of messages in this folder
82	4	unsigned long ulGroupTid	Group tid of folder if it is a group folder.
86	2	unsigned bIsGroup	Specifies if this is a group folder
88	2	unsigned bGroupAttr	a group folders attributes are stored in bGroupAttr so that older versions can't read them.
90	2	short fFoldersVersionNum	version number of folders system 0 = 2.1b DOS Mail, 2.1 WinMail, 2.1beta Mac 1 = ulDeleted, cDeleted fields kept up to date
92	66	char SLACK[66]	

3.2.2. Folder data file format

Folder data files contain a single folder data header followed by zero or more folder data records. Each folder data record contains a single message.

3.2.2.1. Folder data header

offset	size	name	description
0	4	unsigned long ulFirst	offset of first record
4	4	unsigned long ulLast	offset of last record
8	4	unsigned long ulEncryptFlag	If zero, the folder data header is encrypted. If non-zero, the folder data header is not encrypted. Note that the folder data records are never encrypted.
12	4	unsigned long ulDeleted	Total size of deleted records.
16	4	unsigned long cInUse	Number of records in use.
20	4	unsigned long cDeleted	Number of deleted records.
24	25	char szName[25]	Owner of the folder

49	2	int secure	This is a bit field specifying the type of folder. TYPE_MASK 0x0003 one of the following: TYPE_NONE 0 TYPE_PRIVATE 1 TYPE_GROUP 2 TYPE_SHARED 3
51	4	unsigned long value	The contents of this field depends on the type of folder specified in the secure field. if TYPE_PRIVATE->boxkey, if TYPE_GROUP->group_tid, if TYPE_SHARED->owner_boxkey
55	4	unsigned long ulArchiveMagic	magic value for identifying OS files as folders FLD_ARCH_MAGIC = 0x900611FA
59	4	unsigned long ulStartPoint	Current top left corner of folders window
63	2	int fCompressType	for future use to indicate the type of data compression used on an archived folder
65	4	unsigned long ulOpenCount	number of times this folder has been logically opened by Mail Clients (RM 11-20-90)
69	2	short fFolderVersionNum	0 = 2.1b DOS Mail, 2.1 WinMail, 2.1beta Mac 1 = ulDeleted, cDeleted fields kept up to date unique ID for messages, 2.2 functionality
71	58	char SLACK[58]	

3.2.2.2. Folder data records

The ulNext and ulSize fields are used to retrieve the next folder item and the ulMAGIC field is used to authenticate the folder entry. Mail items residing in folders follow FDF_REC records. Each record contains a single message.

offset	size	name	description
0	4	unsigned long ulMAGIC	#define MAGIC 0x8923fd43L
4	4	unsigned long ulNext	offset of next record
8	4	unsigned long ulPrev	offset of previous record
12	4	unsigned long ulSize	Deleted items have size of -1
16	11	char szFrom[11]	Message originator
27	41	char szSubject[41]	Message subject
68	1	char cPriority	Message priority
69	1	char cAttachments	Number of message attachments
70	10	INTERDATE dt	Delivery time of the message
80		FIPS message	A FIPS encoded message.

3.3. The GLB subdirectory

The files in this directory are at the top of the hierarchy of the database and serve a variety of purposes. These files contain the core of the postoffice configuration. The GLB directory also contains error message and other types of files which do not contain postoffice configuration information. These files will only be discussed to a limited extent.

3.3.1. Directory of M:\COURIER2\GLB

ACCESS.GLB	- local user information
ACCESS2.GLB	- extended local user information
ACCESS3.GLB	- remote user dial out information
CONTROL.GLB	- file name generation
GALNETPO.GLB	- global address list
GLOBAL.GLB	- default user configuration information
GROUP.GLB	- list of postoffice groups
GRPMEM.GLB	- list of groups that each user belongs to
MASTER.GLB	- postoffice identification file
NETPO.GLB	- external user addresses
NETWORK.GLB	- network and gateway definitions
TID.GLB	- tuple identifier generation
FLAG.GLB	- used for IPC between External and Admin
MODEM.GLB	- contains local modem configuration
RNETWORK.GLB	- directory for remote users
SERVER.GLB	- dynamic drive connection information
TRANSFIL.GLB	- transparency file for modem connections
WELCOME.GLB	- welcome message
ERRORADM.GLB	- error message file for ADMIN.EXE
ERRORMON.GLB	- error message file for MONITOR.EXE
ERRORMAI.GLB	- error message file for MAIL.EXE
ERRORDYN.GLB	- error message file for DYNADMIN.EXE
RVW.GLB	- contains list of limited directory views
nnnnnnnnn.GLB	- limited directory view file
<modem>.SCR	- compiled modem script file
<modem>.MDM	- uncompiled modem script file

3.3.2. ACCESS.GLB

This file is generally used by user agent programs and contains the users' operating environments. Each user is represented by a single record in this file. ACCESS2.GLB and ACCESS3.GLB are parallel files in the sense that if record n in one file is deleted then it is also deleted in the other file and if record i is for user "joe" in one file then record i is also for user joe in the other files.

In the event of disagreement between replicated fields in ACCESS.GLB, ACCESS2.GLB, or ACCESS3.GLB, the fields contained in ACCESS2.GLB will be considered to be correct.

The structure of the ACCESS.GLB file consists of zero or more records with the following format:

offset	size	name	description
0	2	int ndelete	0 if deleted, non-zero if record is in use.
2	11	char szMailbox[11]	sign on name
13	9	char szAlias[9]	.MBG file prefix
22	9	char npw[9]	user password
31	2	int naccess	Access rights b0: b1: b2:External access b3:Urgent access b4:Delete access b5:Send access b6:Read access b7:Administrator access
33	11	char link[11]	reserved
44	1	char hot_shift	TSR hot key shift state
45	1	char hot_key	TSR hot key scan code
46	1	char hot_disturb	TSR disturb option
47	1	char ed_mode	editor mode
48	1	char remote_config	config bits for remote purpose, eg. pswd
49	9	char unused[9]	not used
58	2	int printerPort	default printer port for DOSMail
60	2	unsigned op_hot_shift	operator program hot key shift state
62	2	unsigned op_hot_key	operator program hot key scan code
64	4	unsigned long gui_opts	Mac and Windows user preferences
68	4	unsigned long ulWhereGrp	offset of users list of groups in GRPMEM.GLB
72	2	int naccess2	Folder access rights. Note that b0 and b1 cannot both be set. b0:create and access private folders b1:access shared folders b2:create and access private folders, access shared folders. b3:create shared folders b4:create group folders b5: b6: b7:
74	2	int list_pref	Address list preference
76	2	int life_reg	regular mail life
78	2	int life_urg	urgent mail life
80	2	int numofhdrs	maximum number of mailbag headers to be loaded.
82	2	int timeout	timeout for mini mail monitor
84	2	int upcount	count of new mail since last signon
86	2	int uncount	count of unread mail
88	2	int key	logical record position of this record
90	2	int lines	lines per page to print

92	2	int autoff	do auto formfeed
94	2	int margin	number of spaces in print margin
96	2	int sortkey	mail header sort key
98	68	char load_path[68]	path to load .MSG files
166	68	char save_path[68]	path to save .MSG files
234	68	char satt_path[68]	path to save attachments
302	68	char view_path[68]	path to view files
370	68	char prin_path[68]	path to print files
438	68	char eatt_path[68]	path to editor attach files
506	68	char incl_path[68]	path to editor include file
574	11	char mail_path[11]	path to server
585	1	char editattr	default editing attribute

Total size = 586 bytes

3.3.3. ACCESS2.GLB

This file is mostly used by external agents to load all information required for successful delivery of mail.

offset	size	name	description
0	2	int ndelete	0 if deleted, non-zero if record is in use.
2	11	char nmailbox[11]	sign on name
13	9	char nalias[9]	.MBG file prefix
22	31	char fullname[31]	users full name
53	2	int naccess	Access rights b0: b1: b2:External access b3:Urgent access b4:Delete access b5:Send access b6:Read access b7:Administrator access
55	4	long tid	tid value as found in admin.nme
59	2	int naccess2	Folder access rights. Note that b0 and b1 cannot both be set. b0:create and access private folders b1:access shared folders b2:create and access private folders, access shared folders. b3:create shared folders b4:create group folders b5: b6: b7:
61	4	long rviewid	remote view id
65	4	char space[4]	not used

Total Size = 69 bytes

3.3.4. ACCESS3.GLB

This file contains additional user agent information as well as dial up information for CALL/REQUEST remote users.

offset	size	name	description
0	2	int ndelete	0 if deleted, non-zero if record is in use.
2	11	char szMailbox[11]	sign on name
13	21	char phone[21]	users phone number
34	6	char extension[6]	users extension
40	15	char room[15]	users office number or room name
55	38	char unused[38]	not used
93	2	int regular	if non-zero then reg. intervals
95	1	char Fill	not used
96	2	DTIME f_times	dialing from what time times table typedef struct { char hour; /* the hour to dial */ char minute; /* the minute to dial */ } DTIME, *PDTIME;
98	2	DTIME t_times	dialing to what time times table typedef struct { char hour; /* the hour to dial */ char minute; /* the minute to dial */ } DTIME, *PDTIME;
100	50	char compose[50]	logfile for compose messages
150	50	char read[50]	logfile for read messages
200	2	int comp_auto	automatic logging
202	2	int read_auto	
204	46	char unused2[46]	automatic logfile for compose messages
250	100	char route2[80+20]	use this field if call/request user
350	2	int bMode	folder stored locally flag
352	68	char szLocalPath [68]	local folder path
420	2	int bCOn	
422	2	int bROn	
424	6	FOLDPTR fpComp	typedef struct foldptr { int fMode ; unsigned long ulWhere ; } FOLDPTR, *PFOLDPTR ;

430	6	FOLDPTR fpRead	typedef struct foldptr { int fMode ; unsigned long ulWhere ; } FOLDPTR, *PFOLDPTR ;
436	4	long lDocRec	only actually used by doc version
440	1	Bullet flag byte	0x80 1 <= > private folder file stored locally 0 <= > private folders stored on post office 0x40 1 <= > the user has run Bullet at least once 0x20 1 <= > private folder file is being moved 0x10 1 <= > private folders were local when move began 0x08 1 <= > the private inbox folder is being shadowed in the mailbag 0 <= > mail is deleted from mailbag as soon as it has been downloaded to the private inbox folder 0x04 unused 0x02 unused 0x01 unused
441	67	Bullet folder path[67]	Path to the Bullet private folder file when it is not stored on the server.
508	4	char space[4]	not used

Total Size = 512 bytes

3.3.5. CONTROL.GLB

This file contains two long integers which are used for generating new, unique file names. One is for generating new mailbag names and one is for generating new mail and attachment names. Since this file has the potential of being the bottleneck for any postoffice, it should only be opened and closed as quickly as possible.

offset	size	name	description
0	4	long nmailbag;	for generating user mailbags file names
4	4	long nletter	for generating mail and attachment file names

3.3.6. GLOBAL.GLB

This file contains default user configuration information which is used when creating new users.

offset	size	name	description
--------	------	------	-------------

0	9	char def_pswd[9]	Default password for new users on this post office. If NULL then passwords are not required
9	2	unsigned access	default access rights for new user
11	1	char hot_shift	TSR hot key shift state: CTRL, ALT, SHIFT
12	1	char hot_key	TSR hot key scan code
13	1	char hot_disturb	TSR disturb option
14	2	int timeout	timeout for mini mail monitor
16	1	char ed_mode	editor mode user prefers to be in
17	2	int life_reg	regular mail life
19	2	int life_urg	urgent mail life
21	2	int numofhdrs	the number of header
23	2	int lines	lines per page to print
25	2	int autoff	do auto formfeed
27	2	int margin	spaces in print margin
29	2	int sortkey	mail header sort key
31	1	char display_mode	display mailbox or fullname
32	68	char def_path[68]	default path to display
100	2	int calendar	calendar defaults, courier admin ignore if calendar module not installed new user gets calendar privileges?
<u>102</u>	<u>2</u>	<u>int access2</u>	<u>Default folders access for new users</u>
<u>104</u>	<u>2</u>	<u>int dos_access</u>	<u>Default to DOS access</u>
<u>106</u>	<u>40</u>	<u>char phnum[40]</u>	<u>Local PostOffice's phone number</u>
<u>146</u>	<u>9</u>	<u>char sxClient[9]</u>	<u>The client (usually admin) that has flag.glb locked open</u>
<u>154</u>	<u>11</u>	<u>char szMailbox</u>	<u>Mailbox name of person running client above</u>
<u>165</u>	<u>31</u>	<u>char szFullname</u>	<u>Full name of person running client</u>
<u>196</u>	<u>315</u>	<u>char unused[315]</u>	<u>not used</u>

3.3.7. GROUP.GLB

The GROUP.GLB file contains a series of records, each of which contains a delete flag, a group name, and a pointer into the ADMIN.GRP file to the file record that is the first member of this group. This record in the ADMIN.GRP file contains the user's name and a pointer to the next member record of this group. Basically a group's members form a chain within the ADMIN.GRP file. The following is the definition for the group.glb file which is defined in the file "ext.h"

offset	size	name	description
0	2	int delflag	
2	4	long tid	unique tuple identifier for this record. the prefix filename of a MEM file which contains all the user mailbags that belongs to this group
6	31	char szRefname[31]	Groups alias name. (what the user sees in their address list)

37	11	char szGrpname[11]	Group address (unique group name for this network and postoffice)
48	2	int grp_type	for postoffice groups only, export option
50	1	char fill	not used

3.3.8. GRPMEM.GLB

The file GRPMEM.GLB contains all the lists of groups that each user belongs to. This file is maintained by the Admin program. It is accessed by Mail on startup/Login New User. This file is used primarily by the folders code for group-shared folders. The File GRPMEM.GLB has the following format:

Group member header

The group member header keeps track of deleted records in the GRPMEM.GLB file. The first four bytes of this file contain a delete chain pointer. This long integer value holds the file position of the first deleted entry. That entry's tid field then contains the file position of the next deleted entry, or NULL if there are no more deleted records. If the delchain pointer is NULL then it means that there are no deleted entries and that the next one to be added should be appended to the end of the file.

offset	size	name	description
0	4	long delchain	file position of the first deleted record in the deleted record chain.

3.3.8.1. Group member records

The group member header is followed by zero or more group member records.

offset	size	name	description
0	2	int delflag	
2	4	long nextmember	next member in the delete chain
6	2	unsigned usedtid	bitmap for each tid in block 1 if used 0 if not
8	2	unsigned status	0 if no more blocks. 1 if there are more blocks
10	48	char SLACK[48]	not used
58	4	unsigned long ulMbgTid	mail bag tid of user this list belongs to
62	2	int num_tids	number of tids in block. This value is less than or equal to 16. When a block is filled with tids a new block is allocated at the end of the file (or from the delete list) and added to the list. The pointer to the start of the list (for each user) will be stored in their access files.
64	64	long grp_tids[16];	tids in this block.

3.3.9. MASTER.GLB

This is the main postoffice identification file which contains information such as serial number, version of postoffice, maximum number of users, gateways installed, etc. This file is not encrypted, but the serial number is stored in a special encrypted format within the file. There is one record in this file which contains the following structure:

MASTER.GLB file, single record file that contains some junk, a copyright notice and the network courier checksummed, encrypted serialization number. It also contains the local network name, the local postoffice name, the local postoffice number, the local postoffice signin ID and the local postoffice sign in password.

```
#define PO_UNASSIGNED 32101 /* post office has not been defined */
```

offset	size	name	description
0	40	char cr[40]	copyright notice
40	30	char po[30]	product name
70	11	char szNetid[11]	local network name
81	11	char szPoid[11]	local postoffice name
92	11	char szSignid[11]	local postoffice signin ID
103	9	char signpswd[9]	local postoffice signin password
112	3	char unused[3]	not used
115	2	int d1	hop count for mail v2.0, default 10
117	10	char md[10]	first five chars used for datagrams
127	5	char fl[5]	not used ("HE>F" for V2.0, "ÈÁ¼Æ" for V1.0)
132	2	unsigned ck	checksum on sn
134	10	char sn[10]	serialization number, 4 digit lot number, and 5 digit serial number
144	2	int d2	max number of users The possible values for the d2 field are: 6 user, 150 user and 500 user. The defines for this are: USERS_UNDEFINED 0x829B USERS_6 6 USERS_150 150 USERS_500 500

146	4	long d3	the Network Courier postoffice type and the modules that are installed. SINGLEPOSTOFFICE 0x14955E5B INTERNETWORK 0x80000000 PCPOSTOFFICE 0x40000000 WORKING_MODEL 0x20000000 The masks for the possible upgrades for the Inter-Network version are defined next (d3 field): REMOTE 0x00000001 PROFS 0x00000002 SNADS 0x00000004 MCI 0x00000008 X400 0x00000010 DOCUMENT 0x00000020 FAX 0x00000040 MHS 0x00000080 SMTP 0x00000100 DEC 0x00000200 UNIX 0x00000400 OV 0x00000800 MS 0x00001000 GAL 0x00002000 DIRSYN 0x00004000
150	1	char d4	The network operating system type can currently be any one of the following: NETOS_UNDEFINED 0x1B NETOS_IBM 0x01 NETOS_NOVELL 0x02
151	9	char szMap[9]	8 used for SNADS po to DGN mapping
160	9	char szOvmap[9]	OV map info for DOMAIN name

169	1	char d5	The d5 field will be used for the server version abort. It is currently set to 0 by the install program. This needs to be implemented in every PCMail 3.0 "user agent" that goes in the PC Mail box. These are specifically: - Admin - Dynadmin (sp?) - Mac client - DOS client - Micro - Monitor The intention here is to have a mechanism that we can use to prevent any 3.0 client from running against a future database (i.e. 4.0) in order to prevent database corruption. Define this constant in ext.h (where MASTER is defined): <pre>#define DB20_COMPATIBLE 0 /* 2.0 compatible database */</pre> On startup, check d5 to make sure it is zero. If not, abort with an error message. ie: <pre>if (master.d5 != DB20_COMPATIBLE) exit("This program is incompatible with the postoffice.");</pre>
-----	---	---------	--

170	1	char ver	product release version number These are the values that the ver field can have: <pre>#define DBVER_20 0 /* 2.0 release */ #define DBVER_21 1 /* 2.1 release */ #define DBVER_30 2 /* 3.0 release */</pre> This field will only be of immediate use for debugging utilities. It will be incremented by one on each successive release of PCMail.
171	2	unsigned winmail_id;	version id for windows mail
173	3	char fill[3]	not used

Total size is 176 bytes.

3.3.10. NETPO.GLB

The NETPO.GLB file is used to keep a copy of all external Network Courier user addresses that are added to either the postoffice address list or any user's personal address list. This reduces the number of files that need to be opened and closed when it comes time to resolve these external addresses.

There is one NETPOHEAD record followed by any number of NETPO records.

NetPo Header

The NetPO header keeps track of deleted records in the NETPO.GLB file. The first four bytes of this file contain a delete chain pointer. This long integer value holds the file position of the first deleted entry. That entry's tid field then contains the file position of the next deleted entry, or NULL if there are no more deleted records. If the delchain pointer is NULL then it means that there are no deleted entries and that the next one to be added should be appended to the end of the file.

offset	size	name	description
0	4	long delchain	file position of the first deleted record in the deleted record chain.

3.3.10.1. NetPo Records

The NetPo header is followed by zero or more NetPo records.

offset	size	name	description
0	2	int delflag	
2	4	long tid	unique tuple identifier for this record
6	11	char szNetname[11]	
17	11	char szPoname[11]	
28	11	char szMailbox[11]	
39	4	long mailbag	

3.3.11. NETWORK.GLB

This file contains two types of record definitions, DNETWORK and DGATENET. Both of these records are the same size, so the records are generally read in as DNETWORK records and cast as DGATENET records when necessary.

3.3.11.1. DNETWORK structure:

The DNETWORK structure defines a Network Courier network name and contains the name of the XTN file containing individual postoffice information.

offset	size	name	description
0	1	char del_flag	record deleted or not
1	1	char serv_type	service type (CSI, MCI, PROFS)
2	11	char szNet_name[11]	external service name
13	9	char szDat_file[9]	route info datafile
22	100	char future[100]	not used

3.3.11.2. DGATENET structure:

The DGATENET structure defines a gateway post office. Network names are defined by the Admin program and a new record is added to this file every time a new network name is defined. Gateway names are defined and added to this file at installation time. Does not need to create .xtn file for gateway postoffice.

offset	size	name	description
0	1	char del_flag	record deleted or not
1	1	char serv_type	service type (CSI, MCI, PROFS)
2	11	char szNet_name[11]	external service name
13	9	char szDat_file[9]	route info datafile
22	11	char po_name	name of service or post office
33	9	char szMailbag[9]	external mailbag file pointer
42	2	int mm_size	maximum mail item size allowed to send
44	2	int regular	if non-zero, then regular intervals
46	2	int retries	maximum number of mailing retries
48	2	int reg_delay	failed connection retry for regular mail
50	2	int urg_delay	failed connection retry for urgent mail
52	1	char conf_opt	bit1: reg. mail confirmation option bit 2: two way mail flag
53	1	char cur_month	current month mtd stats for 1..12
54	1	char cur_year	current year stats for 86, 87, ...
55	2	unsigned mtd_num_sent	month to date mail sent
57	2	unsigned ytd_num_sent	year to date mail sent
59	2	unsigned p_mtd_unused	not used
61	2	unsigned p_ytd_unused	not used
63	2	unsigned e_mtd_unused	not used
65	2	unsigned e_ytd_unused	not used
67	4	long e_mtd_cents	external month to date mail cost

71	4	long e_ytd_cents	external year to date mail cost
75	1	char indirect	0=direct 1=indirect
76	11	char szInd_net[11]	forward through external service name
87	11	char szInd_post[11]	forward through external postoffice
98	9	char szInd_mailbag[9]	forward through external mailbag
107	2	int ind_count	the # of indirects that point to this PO
109	1	char connect	0=connection via drive 1=connection via modem
110	12	char future[12]	not used

3.3.12. TID.GLB

The Network Courier maintains unique tuple identifiers in several file record structures. These tuple identifiers are used to guarantee the authenticity of each user name. For example, if John Smith has the mailbox name JohnS and eventually gets replaced with a new user named JohnS, the tuple identifier gets changed with the deletion and recreation of the mailbox name. Any users who had the old JohnS in their address list would no longer have a valid address definition since the tuple identifiers (or "tid's") would not match.

The file which is used for generating new tid's is TID.GLB and contains a single long integer.

3.3.13. Gateway address lists

The formats of the gateway address lists are defined by the TPL files. However, the entries in the gateway address list are packed.

3.4. The GRP subdirectory

The first record in the ADMIN.GRP file contains a record that points to the first record in a chain of records in this file that may be reused when a new group is created. This chain is added to when an existing group is deleted.

The record in the ADMIN.GRP file contains the user's name and a pointer to the next member record of this group.

Individual users may also define their own groups. The name of the group is in the user's nnnnnnnn.NME file and the record for that group contains a file position into the user's nnnnnnnn.GRP file. The members of the group are chained together the same way that Admin defined groups are. When a group is deleted, the chain of group members gets added to the delete chain. If a single member is deleted, he is removed from the group chain and added to the delete chain.

Note that group members are chained together by record numbers or "keys" rather than by file position numbers or "bytepos".

offset	size	name	description
0	11	char szMemname[11]	member name
11	11	char szPostname[11]	postoffice name of member
22	2	unsigned key	next member pointer
24	2	int count1	
26	2	int count2	

3.5. The HLP subdirectory

This directory contains help files for all Network Courier programs. These files are preprocessed with the MAKEHELP.EXE program. The format of these processed help files will not be discussed here.

3.6. The INF subdirectory

The INF file contains template or "about" information for every user defined in a USR file. The USR file is the user list for external Network Courier postoffices and has the same filename prefix as the postoffice's MBG file. The INF file also has this same filename prefix. There may also be an ADMIN.INF file if the local postoffice administrator has defined a custom template.

External postoffice INF files only exist if another postoffice administrator exports templates.

The record format of the INF file is defined by the associated TPL file. There is always an associated TPL file with every INF file.

The first four bytes of this file contain a delete chain pointer. This long integer value holds the file position of the first deleted entry. That entry's tid field then contains the file position of the next deleted entry, or NULL if there are no more deleted records. If the delchain pointer is NULL then it means that there are no deleted entries and that the next one to be added should be appended to the end of the file.

```
{
  int delflag;
  long tid; /* unique tuple identifier for this record */
  char record[cbRecord];
}
```

Each record consists of a sequence of fields defined by the associated TPL file. Each field contains an internal field separator followed by the field data.

3.7. The KEY subdirectory

The KEY file contains information such as the number of items of newmail since the user last logged in, the amount of unread mail in the user's mailbag, and a bitmap of deleted records in the corresponding MBG file. The bitmap is used so that when mail is sent to a user, a vacant record position in the user's MBG file can be found directly by examining the bitmap rather than having to sequentially process the MBG file looking for deleted records.

The mailbag (MBG) file which corresponds to a particular KEY file has the same filename prefix as the KEY file.eg 12345678.MBG <===> 12345678.KEY - user 12345678's mailbag

offset	size	name	description
0	2	int upcount	new mail since last sign on or Update
2	2	int rcount	count of all unread mail
4	512	char bitmap[4096/8]	bitmap of deleted headers in the corresponding .MBG file.
516	2	int mon_mess	message from mail to monitor
518	2	int mail_mess	message from monitor to mail
520	2	int lkey	used so that we know which message last new one was
522		char space[UNUSEDSPACESIZE-2]	

3.8. The LOG subdirectory

The LOG directory contains log files which are maintained by the External program and by gateway Message Transfer Agents. The files that are maintained by the External program are:

```
CALLIN.LOG - record of dial-in sessions
CALLOUT.LOG - record of dial-out attempts
RECV.LOG - record of received mail
SENT.LOG - record of sent mail
SESSION.LOG - transcript of External session
SYSTEM.LOG - contains system error messages
```

3.9. The MAI subdirectory

The MAI directory contains 16 subdirectories. These are named as follows and contain the following types of filenames:

```
MA0 - nnnnnnn0.MAI
MA1 - nnnnnnn1.MAI
MA2 - nnnnnnn2.MAI
MA3 - nnnnnnn3.MAI
MA4 - nnnnnnn4.MAI
MA5 - nnnnnnn5.MAI
MA6 - nnnnnnn6.MAI
MA7 - nnnnnnn7.MAI
MA8 - nnnnnnn8.MAI
MA9 - nnnnnnn9.MAI
MAA - nnnnnnnA.MAI
MAB - nnnnnnnB.MAI
MAC - nnnnnnnC.MAI
MAD - nnnnnnnD.MAI
MAE - nnnnnnnE.MAI
```

MAF - nnnnnnnF.MAI

Mail files are fully encrypted and composed of two parts. The first part is the "system" information or mail header. This section contains most of the FIPS fields which are described later in this document. The second part is the TEXT field which is the part of the message that the user actually sees. The TEXT field is represented as a single FIPS field.

3.10. The MBG subdirectory

The MBG (mailbag) file contains records which serve two purposes. First, a non-deleted record acts as an index pointer to an item of mail. Thus, a single mail file can be pointed to by many mailbag records. Each of these mailbag records must reside in separate MBG files since two records in a single mailbag can never point to the same item of mail.

The second use for MBG records is to contain summary information about the particular mail item. This information is used by the user agent programs to display a list of mail headers in the opening display of the users mail. The mail file itself does not have to be read until the user selects it for processing (eg reading, deleting, etc).

There is no delete flag in the MBG record structure. Deleted status is maintained in the KEY file which has the same filename prefix as the MBG file. See The KEY Subdirectory for more information.

There is a single MBG file for every local user of the format nnnnnnnn.MBG, plus a single MBG file for every external postoffice or gateway that is defined, plus an INQUEUE.MBG file which is used to receive any mail from external sources. Mail is delivered from INQUEUE.MBG to individual mailbags via the Mailer process.

The structure of user mailbag file (.MBG type) for letter headers

offset	size	name	description
0	2	ispace	not used
2	11	char szFrom[11]	message originator
13	41	char szSubject[41]	message subject
54	2	char szPriority[2]	message priority or message type 1:Low priority 2,3,4:Normal priority 5: High priority, with expedited delivery. R: Registered (receipt notification requested) C: Confirmation of receipt. (Receipt notification) T: Telephone message F: File transfer
56	10	INTERDATE date	message delivery time {int minute, hour, day, month, year;}

66	1	char status	This is a bit field which specifies if the message has been read, or a receipt notification has been sent. b0: Registered ? b1: Message has been read.
67	9	szBox_key[9]	raw file name of message
76	2	int attachments	number of attachments included
78	4	long total_size	total size of mail file and attachments
82	1	char type	reserved bitfield b0:from remote user, enables group expansion.
83	4	<u>Bullet 32-bit ID of the message in the MMF file</u>	<u>Mark a message as having been downloaded into the MMF file, when the user has requested that Bullet not delete their mail from the mailbag right after downloading; this mode goes under the sonorous name of "inbox shadowing."</u>
87	1	char retries	number of unsuccessful attempts to send
88	2	int hours_useful	useful mail life
90	2	int iLockId	The process number of a process that has a lock on this reference to the mail item, or zero if no processes have the current reference locked.
92	4	char compressionType[4]	compression algorithm identifier
96	4	char encryptionType[4]	encryption algorithm identifier
100	8	char encryptionKey[8]	encryption key
108	8	char space[8]	not used

3.11. The MEM subdirectory

This directory contains the group member files for every group that is defined. Each record is a long integer which represents a user mailbag number. These files are only created for Admin defined groups (ie postoffice groups) and are used for quick access by Mailer for resolving groups. This file is a max size of 500 (users) x 4 (bytes) = 2000 bytes

3.12. The NME subdirectory

This directory contains the postoffice address list and the personal address lists. The postoffice address list (POL) is defined in the file ADMIN.NME and personal address lists (PAL's) are defined in files of the form nnnnnnnn.NME where nnnnnnnn is the filename prefix of the user's MBG file.

GAL.NME contains the global address list.

The record structure of the POL and PAL defines the fullname or alias of the user along with a file position pointer into one of the following files:

ACCESS.GLB - local users

GROUP.GLB - local groups
 NETPO.GLB - external Network Courier users
 GALNETPO.GLB - external Network Courier users for
 global address list.
 xxxx.GLB - for gateway addresses

The file that the bytepos field refers to depends on the type field (local, group, external, gateway).

3.12.1. Alias Record

offset	size	name	description
0	2	int type	
2	31	char szRefname[31]	
33	4	long tid	unique tuple identifier for this record
37	4	long bytepos	
41	4	long fill	

3.13. The P1 subdirectory

The P1 directory is used for two types of files. First, "envelope" or "P1" (a term borrowed from X.400) files are stored here temporarily while mail is being sent or received. P1 files are in the same format as MAI files except that P1 files have no TEXT field. The TEXT field of a message is the part that the user actually sees.

P1 files also have no CC field. The original TO and CC fields are merged and filtered to determine the proper subset of addresses to mail to. The final address list is put in the TO field of the message.

P1 files have the format nnnnnnnn.P1 or nnnnnnnn.Pid where nnnnnnnn is the filename of the matching MAI file and "id" is the process id of the creator of the file. Process id's are discussed in greater detail in the section on Mailer.

The P1 directory is also used for temporary storage of "mbg_busy" files. The function mbg_busy() uses file creation and deletion as a semaphore mechanism to prevent against simultaneous access to a single mailbag for mail collection by multiple external agents. "External agents" means any gateway MTA or the External program. This process is discussed in greater detail in the sections on Mailer and Collector.

The files created by mbg_busy() are zero length files which have the format nnnnnnnn.Oid where nnnnnnnn is the name of the MBG file being locked and "id" is the process id of the external agent which currently has the mailbag locked.

It is important to use this form of mailbag locking rather than file sharing modes because the Mail program and other user agents require write access to all mailbags as often as possible. Using file sharing modes as a locking mechanism would prevent against this.

The mbg_busy() algorithm will likely be replaced in a future version of The Network Courier as it is recognized as being inefficient.

3.14. The TPL subdirectory

This directory contains template files. There are some predefined templates which are used for data entry purposes, and there may also be an ADMIN.TPL and several nnnnnnnn.TPL files. The existence of these files will depend on whether administrators have defined custom templates.

The contents of the template file define the record size of the associated INF file. So once an INF file is created, the TPL file cannot be arbitrarily changed. There are, however, utilities to allow modification of templates.

```
delimiter ::= '~'
terminator ::= '\r'
```

```
TemplateField ::= SEQUENCE {
    prompt string
    delimiter
    start number
    delimiter
    width number
    delimiter
    edit string
    delimiter
    data string
    delimiter
    internal string
    terminator
}
```

3.14.1. Prompt

This is a text label that describes the field. The prompt appears to the left of the text entry field

3.14.2. Start

This field is used to layout an entry field. It specifies the character position from the left margin where the entry field should be displayed.

3.14.3. Width

The number of characters in the field.

3.14.4. Edit

The permissible characters which may be entered into the string. All characters are assumed to be in the codepage 850 character set.

	Description
U	Uppercase characters
N	Numeric characters
L	Lowercase characters
P	Punctuation characters
A	Any characters

3.14.5. Internal

This is an internal preamble which will be stored in the INF record before the field data.

3.15. The USR subdirectory

This file is the address list of users and groups for a particular postoffice. The postoffice that this list represents is an external defined postoffice. If this file exists, it's prefix filename will match the external postoffice's MBG file name. This file can be created or updated using the Admin program or can be created or replaced by the External program when it receives an exported user list from this postoffice.

/* structure for exporting and importing userlists */

offset	size	name	description
0	11	char szMailbox[11]	sign on name
11	31	char szFullname[30+1]	users full name
34	1	char Fill	byte alignment filler for MSC

3.16. The XTN subdirectory

For each network that is defined in the NETWORK.GLB file there is an associated XTN file. This file is a series of records, where each record defines a different postoffice at this network. The purpose of this file is to provide routing, naming and signon information for external services/postoffices

Postoffice records also contain connection information, statistics and various configuration options for this postoffice. Each postoffice record contains a file pointer reference to its mailbag.

When an address list of users is created for this postoffice there will also be a USR file and possibly TPL and INF files with the same file prefix as the MBG file.

network Datafiles (nnnnnnnn.xtn)

offset	size	name	description
0	1	char del_flag	record deleted or not
1	11	char po_name[11]	name of service or post office
20	9	char mailbag[]	external mailbag file pointer
29	2	int mm_size	maximum mail item size allowed to send
31	2	int regular	if non-zero, then regular intervals
33	2	int frst_cost	first minute dialing charge in cents
35	2	int addl_cost	additional minutes dialing charge
37	1	char Fill1	not used
38	12	DTIME f_times[6]	dialing from what time times table typedef struct { char hour; /* the hour to dial */ char minute; /* the minute to dial */ } DTIME, *PDTIME; #define T_UNINIT 255 /* uninitialized value for hour/minute */
50	12	DTIME t_times[6]	dialing to what time times table typedef struct { char hour; /* the hour to dial */ char minute; /* the minute to dial */ } DTIME, *PDTIME; #define T_UNINIT 255 /* uninitialized value for hour/minute */
62	12	int f_cost[6]	first minute dialing charge in cents
74	12	int a_cost[6]	additional minutes dialing charge
86	2	int retries	maximum number of mailing retries
88	2	int reg_delay	failed connection retry for regular mail
90	2	int urg_delay	failed connection retry for urgent mail
92	1	char conf_opt	bit field: bit 1: do not allow registered returns bit 2: allow two way mail
93	9	char filler[9]	not used
102	1	char cur_month	current month mtd stats for 1..12
103	1	char cur_year	current year stats for 86, 87 etc
104	2	unsigned mtd_num_sent	month to date mail sent
106	2	unsigned ytd_num_sent	year to date mail sent
108	2	unsigned p_mtd_unused	not used
110	2	unsigned p_ytd_unused	not used
112	2	unsigned e_mtd_unused	not used
114	2	unsigned e_ytd_unused	not used
116	4	long e_mtd_cents	external month to date mail cost
120	4	long e_ytd_cents	external year to date mail cost
124	501	char route[500+1]	buffer for dialing route definition
625	1	char indirect	direct or indirect
626	11	char szInd_net[11]	forward through external service name

637	11	char szInd_post[11]	forward through external postoffice
648	9	char szInd_mailbag[9]	forward through external mailbag
657	2	int ind_count	the # of indirects that point to this PO
659	1	char connect	connection via 0=drive or 1=modem
660	9	char map[9]	used for SNADS po to DGN mapping
669	36	char future[36]	not used
705	1	char Fill2	not used

NLS String Support

Outline

The following files make up the NLS string support:

NLSCASE.C	tables and functions for upper/lower case translation
NLSCOMP.C	comparison functions
NLSINT1.C	primary international comparison table
NLSINT2.C	case insensitive secondary international comparison table
NLSINT3.C	case sensitive secondary international comparison table
NLSIS.C letter, etc.)	tables and functions for determining character type (number,
NLSLOAD.C	functions for loading external comparison tables
NLSSCAN1.C	primary Scandinavian comparison table
NLSSCAN2.C	case insensitive secondary international comparison table
NLSSCAN3.C	case sensitive secondary international comparison table
NLSTRANS.C	tables and functions for cope page 850 to ANSI translation

The NLSINT1.C, NLSINT2.C, NLSINT3.C, NLSSCAN1.C, NLSSCAN2.C and NLSSCAN3.C are included in NLSCOMP.C and are not separately compiled.

NLSCASE.C

NLSCASE.C provides tables and functions for upper/lower case translation.

NLSCASE.C has two tables, one for upper to lower case conversion and one for lower to upper case conversion. Both of these tables are for code page 850. Each case conversion table provides the lower to upper case translation for the upper 128 characters. The lower 128 characters (a-z) are converted with the usual subtraction.

Tables are not needed for ANSI, since both the lower 128 characters (a-z) and the upper 128 characters can be converted with a subtraction. The only exceptions are the lower case to upper case translations of:

ÿ	to	Y
ä	to	A
ö	to	O

Naturally, once these three characters have been converted to upper case, they cannot be converted back to the original lower case character.

For code page 437, the Network Courier library functions are used (there is no table in this case).

Entry Points

nls_CharToUpper	converts a single character to upper case
nls_StrToUpper	converts a string in place to upper case
nls_CharToLower	convert a single character to lower case
nls_StrToLower	converts a string in place to lower case

Compiler Switches

Compile this file with the following switches to get the functions to use the desired character set.

<none>	Code Page 437 (American)
CP850	Code Page 850 (International)
ANSI	ANSI Character Set

NLSCOMP.C

NLSCOMP.C provides comparison tables and functions.

There are three tables used for comparison. The first table is the Primary Weight table. This table differentiates between different characters. A, a and ā are all considered the same character in this table, whereas a and b are different characters. The second and third tables are the Secondary Weight tables. There is one Secondary Weight table for case sensitive comparisons and one for case insensitive comparisons. Both are diacritic sensitive. These tables differentiate between different forms of the same character. In the case sensitive table, A, a and ā are considered to be different. In the case insensitive table, A and a are considered to be the same but a and ā or A and ā are considered to be different.

Each of the tables described above is available in two different sort orders: International and Scandinavian.

Each of the tables described above is available for both code page 850 and for ANSI.

All told there are then 12 different tables. The following table shows where each of these tables resides. The files contain only the tables and are included by NLSCOMP.C (these files are not separately compiled). Each of the files contains both a code page 850 and a ANSI version of the table.

Table Name	File Name

International Primary Weight Table	NLSINT1.C
International Secondary Weight Table (Case Sensitive)	NLSINT2.C
International Secondary Weight Table (Case Insensitive)	NLSINT3.C

Scandinavian Primary Weight Table	NLSINT1.C
Scandinavian Secondary Weight Table (Case Sensitive)	NLSINT2.C
Scandinavian Secondary Weight Table (Case Insensitive)	NLSINT3.C

When a comparison is done, each letter of the two strings are compared in turn. If two letters are found to be different with regard to the Primary Weight table, then the sort order is determined from these two letters. If two letters are found to be the same with regard to the Primary Weight table, but different with regard to the Secondary Weight table, this fact is remembered, but the sort continues (only the first Secondary Weight table difference is remembered; once one is found, all further Secondary Weight differences are ignored). If both strings are fully compared, and no Primary Weight table difference was found, then the remembered Secondary Weight table value is used to determine the sort order. If no Primary Weight of Secondary Weight differences are found, the strings are, of course, equal.

This two table method is used to make sure that correct sorting order is maintained. The following three strings should sort as follows:

ago
agö
agone

If the ö character were considered to be a separate character, then the second word would sort after the third (ö after o) - which is of course incorrect. The two level sort ensures the above order by first ignoring the diacritic character and sorting only by actual letter differences. But when two words are the same except for a diacritic letter, then a sort order is still enforced. Cases are treated the same as diacritics (that is, a change to the base character requiring a secondary level sort).

Entry Points

nls_CharComp	compares two characters (case sensitive)
nls_StrComp	compares two strings (case sensitive)
nls_CharInComp	compares two characters (case insensitive)
nls_StrInComp	compares two strings (case insensitive)

Compiler Switches

Compile this file with the following switches to get the functions to use the desired character set.

<none>	Code Page 437 (American)
CP850	Code Page 850 (International)
ANSI	ANSI Character Set

With the CP850 and ANSI switches, the SCAND switch may also be used to obtain Scandinavian sorting (ie. use both CP850 and SCAND switches or both ANSI and SCAND switches). Without the SCAND switch, international (Z) sorting will be used.

NLSIS.C

NLSIS.C provides tables and functions for determining character type (upper case, lower case, digit, punctuation or graphic).

NLSIS.C has two tables, one for code page 850 and one for ANSI. Which one is used depends on the selected compiler switch. Each table has one value for each character, defining which set it belongs to. The values that may be assigned to each character are:

NLS_UPPER	upper case
NLS_LOWER	lower case
NLS_DIGIT	digit
NLS_PUNC	punctuation
NLS_GRAPH	graphic
NLS_SPACE	space
NLS_OTHER	other

For code page 437, the Network Courier library functions are used (there is no table in this case).

Entry Points

nls_IsUpper	determines whether a character is upper case
nls_IsLower	determines whether a character is lower case
nls_IsDigit	determines whether a character is a numeric
nls_IsPunct	determines whether a character is punctuation
nls_IsGraph	determines whether a character is graphic

Compiler Switches

Compile this file with the following switches to get the functions to use the desired character set.

<none>	Code Page 437 (American)
CP850	Code Page 850 (International)
ANSI	ANSI Character Set

NLSTRANS.C

NLSTRANS.C provides tables and functions translation between code page 850 and ANSI.

There are two tables in NLSTRANS.C, one for code page 850 to ANSI translation and one for ANSI to code page 850 translation.

Since the lower 128 characters of both character sets is the same, these tables only provide the character translation for the upper 128 characters. All code page 850 characters that exist in the ANSI set are translated one to one. Those code page 850 characters that do not exist in the ANSI set are translated to the undefined characters in the ANSI set on a one to one basis. This applies mostly to the graphics characters provided in code page 850. When translated back to code page 850, the graphics characters will be retained (essentially, this causes the officially "undefined" characters in the ANSI set to represent the graphics characters of code page 850).

Because the translation is done solely on a one to one basis, ANSI can represent the full range of code page 850 values (even if the actual characters themselves are not really defined in ANSI and cannot be seen). Therefore, no information is lost when code page 850 to ANSI to code page 850 translation is done (or ANSI to code page 850 to ANSI translation).

There are several special translations:

Character	Code Page 850	ANSI
Multiplication sign	9E	D7
Division sign	F6	F7
Graphic character	CF	A4

For the first two (multiplication and division signs) the official ANSI character set does not have these characters, but the ANSI set that Microsoft is using in Windows has these two characters in the positions given above. The third character translation is where there is a graphics symbol in ANSI that does not exist in code page 850. In this case, a vaguely similar graphic has been chosen in the code page 850 set.

Entry Points

nls_Cp850ToAnsi	converts a string in place from code page 850 to the ANSI character set.
nls_AnsiToCp850	converts a string in place from the ANSI character set to code page 850.

Compiler Switches

Compile this file with the following switches to get the functions to use the desired character set.

<none>	Code Page 437 (American)
CP850	Code Page 850 (International)
ANSI	ANSI Character Set

The translation functions are compiled only if the CP850 or the ANSI switch has been selected (ie. if you're using code page 437, these translation functions will not be included).

NLSLOAD.C

NLSLOAD.C provides functions for loading external comparison tables. The desired tables must exist in either ANSISORT.GLB or C850SORT.GLB in the GLB directory of the mail data base. If NLSLOAD.C is compiled with the ANSI switch, the ANSISORT.GLB file will be used, if it is compiled with the CP850 switch, then the C850SORT.GLB file will be used. See the External Comparison Tables section for more information on the creation of these GLB files.

Entry Points

nls_LoadTable	loads the external comparison tables
---------------	--------------------------------------

Compiler Switches

Compile this file with the following switches to get the functions to use the desired character set.

<none>	Code Page 437 (American)
CP850	Code Page 850 (International)
ANSI	ANSI Character Set

The table load functions are compiled only if the CP850 or the ANSI switch has been selected (ie. if you're using code page 437, the table load functions will not be included).

External Comparison Tables

A mail data base has two files in the GLB directory that contain external comparison tables: ANSISORT.GLB and C850SORT.GLB. The ANSISORT.GLB comparison file may contain either the International or Scandinavian (not both at the same time) ANSI comparison tables. The C850SORT.GLB comparison file may contain either the International or Scandinavian (not both at the same time) code page 850 comparison tables.

Comparison File Format

The format of the comparison GLB files (they both have the same format) is as follows:

int	Table Identifier - indicates what the table is: 1 = Code page 850 Scandinavian 2 = ANSI Scandinavian 3 = Code page 850 International 4 = ANSI international
char[25]	Table Name - gives the name of the table: "Scandinavian CP850" "Scandinavian ANSI"

	"International CP850"
	"International ANSI"
char[256]	Primary Weight Table
char[256]	Secondary Weight Table (Case Sensitive)
char[256]	Secondary Weight Table (Case Insensitive)

Comparison File Creation

To create the comparison files (ANSISORT.GLB and C850SORT.GLB), the TABLE program is used.

Compile the TABLE.C file with the following switches to get the TABLE program to use the desired character set:

CP850	Code Page 850
ANSI	ANSI Character Set

Compile the TABLE.C file with the following switches to get the TABLE program to use the desired sort order:

<none>	International sort order
SCAND	Scandinavian sort order.

For example, to be able to create the ANSI Scandinavian comparison file, compile TABLE.C with the ANSI and SCAND switches. Then run TABLE.EXE. The output will be ANSISORT.GLB with ANSI Scandinavian tables.

Because, the ANSISORT.GLB file could be either for Scandinavian or International sort orders (ie. the one file could have one of two different sets of contents), we keep copies of each of the sort orders in secondary files ANSIINT.GLB and ANSISCAN.GLB. Either of these two files can be copied to ANSISORT.GLB to change the sort order. The same applies to C850SORT.GLB (its secondary files are C850INT.GLB and C850SCAN.GLB).

4. FIPS Message Format

The Network Courier mail file format is based on FIPS PUB 98 (Federal Information Processing Standards Publication 98, Message Format for Computer-Based Message Systems). This format is a subset of the CCITT X.400 standard for Interpersonal Messaging.

Basically a message can be thought of being composed of triples of the form:

Field Identifier, Length Specifier, Data

All data elements consist of a series of components.

Identifier octet (identifying particular type of data element)

Length Code (specifying number of octets that appear following it in a data element.)

Qualifier

Data Element Contents (containing actual data of the data element)

4.1. Identifier

The identifier octet is a numeric code containing information that identifies a data element. It is always the first component in a data element. The field Identifier is one byte in length and must be one of the following:

Identifier	Description	Type	Qualifier
0x02	ASCII String	Primitive	No
0x20	Integer	Primitive	No
0x28	Date	Constructor	No
0x4C	Field	Constructor	Yes
0x4D	Message	Constructor	No
0x7F	Vendor-defined field	Constructor or primitive	Yes
0x0A	Sequence	Constructor	No
0x0B	Set	Constructor	No
0x08	Boolean	Primitive	No

The second most significant bit of the identifier octet signifies whether or not the data element has a Qualifier. If the bit is set to 1, then the data element has a qualifier (or opcode). Otherwise, the data element does not have a Qualifier. Note that the Message (4D) identifier is an exception to this rule. Although it has the second most significant bit set, it does not have a qualifier.

4.2. Length

The Data can be either another field, a series of fields, or an opcode.

The length specifier specifies the number of bytes in the data portion of the field. The length specifier may require multiple bytes to specify the length. If the first byte of the length specifier has the MSB set, the lower four bits of this byte indicate the number of following bytes that will specify the message length. Otherwise, the length byte specifies the length of the field.

The length code component immediately follows the identifier octet. It is present in every data element. The length code indicates the number of octets following it in a data element (excluding the identifier octet and the length code itself). Length Codes appear in 2 formats: short and long.

A short length code is one octet long. Its most significant bit is set to 0 and its value is in the range 0 through 127.

A long length code is at least two octets long. The first octet always has its most significant bit set to 1. The other seven bits of this octet encode the number of octets in the rest of the component. The actual value begins in the next octet.

The Length Specifier specifies how many bytes follow and can be a variable number of bytes in length. If the first byte of the length specifier contains a 1 in the leftmost position (ie 0x80 or higher) then the lower four bits of this byte indicate the number of following bytes that will specify the message length (max 16). Otherwise the length byte specifies the length of the field.

4.3. Data Elements

4.3.1. Primitive Data Elements

A primitive data element contains a basic item of information; it is not composed of other data elements. The most commonly used primitive data elements are Integer and ASCII-String.

4.3.1.1. ASCII-String

This data element contains a series of ASCII characters. All Network Courier strings are expected to use the IBM code page 850 character set.

```

02    05    hello
|      |      |
|      |      | data (text line)
|      |      | length of string
|      |      | Data Element (ASCII string)

```

4.3.1.2. Integer

This data element contains a two's complement integer of variable length, high order octet first. It is recommended that the data element be either 2 or 4 octets long whenever possible. The following example shows the encoding of 1 as a short integer.

```

20    02    00    01
|      |      |      |
|      |      |      | data (short integer value)
|      |      |      | length of data
|      |      |      | Data Element (integer)

```

4.3.1.3. Boolean

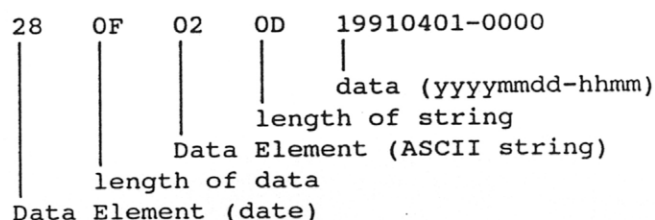
This data element has two possible values, true and false. Not supported by Network Courier 2.1.

4.3.2. Constructor Data Elements

Constructor data elements contain one or more data elements. The most general form of a constructor is a Sequence or Set, since both Sequences and Sets may contain any data element.

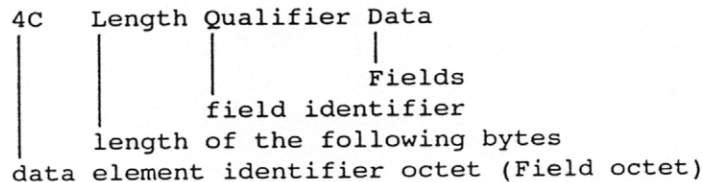
4.3.2.1. Date

This data element contains an ASCII-String data element, which is a representation of a date and time. The use of time and time zone is optional. It is recommended that numeric offsets be used to indicate time zone.



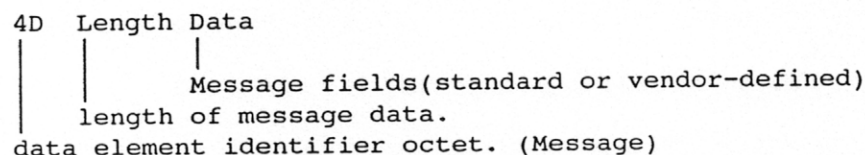
4.3.2.2. Field

This data element contains a field identifier and contents. The Field Identifier indicates which specific field is being represented. The Contents portion of a particular field in a message must contain at least one data element.



4.3.2.3. Message

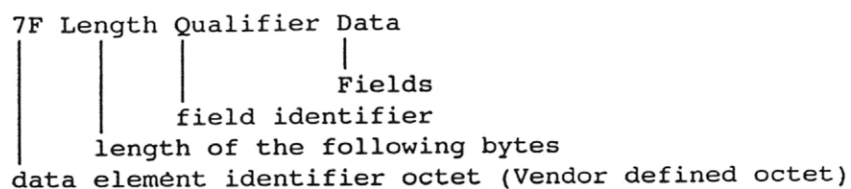
The prefix of the entire message is the MESSAGE field. The prefix is as follows:



A Message data element is a constructor. It may contain only Field data elements, other Message data elements, or encrypted or data compressed forms of these elements. A Field data element can contain any data element. It also indicates which specific field is being represented. The contents of some fields are restricted to a single type of data element.

4.3.2.4. Vendor-Defined

This data element represents Vendor-Defined data elements. A Qualifier component extends the encoding space for identifiers.



4.3.2.5. Sequence

This data element contains any series of data elements. Sequence differs from Set in that the elements must be considered an ordered sequence. Not support by Network Courier 2.1.

4.3.2.6. Set

This data element contains any series of data elements with no ordering of the elements implied. Although the data elements contained in a Set must be stored sequentially. Not supported by Network Courier 2.1.

4.4. Message Fields

A mail file is composed of fields. Each field is composed of a field type, field length and data. Fields may either be as specified in FIPS PUB 98 or Vendor defined fields. A Vendor defined field is a field that may be defined for your own purposes. The Network Courier uses several Vendor defined fields. The following table specifies the fields that a Network Courier mail message is composed of and the order that these fields generally appear in a mail message.

Field name	Field type
Message format	Vendor-defined
From	Vendor-defined
To	Standard
Cc	Standard
Date	Standard
Priority	Standard
Subject	Standard
Attachments	Vendor-defined
Text	Standard

Defining the FIPS standard field identifiers

Opcode	Description
0x01	FROM
0x02	DATE
0x04	TEXT
0x05	TO
0x06	CC
0x07	SUBJECT
0x18	PRIORITY
0x30	COPIES

0x31	ATTACHMENTS
0x32	MFORMAT - Message format
0x33	HOPCOUNT keeps count of number of postoffice hops
0x34	HOPTRACE trace info on postoffice hops
0x35	FOLDATTACH folders attachment data
0x36	TEXTATTR text attributes (v2.1)
0x37	TEXTBORDER text border (v2.1)
0x38	NLSTAG national language support

4.4.1. ATTACHMENTS opcode

```

7F Length 31 02 Length1 file1 02 Length2 file2 etc.
|           |           |           |           |
|           |           |           |           | data (attachment structure)
|           |           |           |           | length of data
|           |           |           |           | Data Element (ASCII string)
|           |           |           |           | field (ATTACHMENTS)
|           |           |           |           | length of the following bytes
|           |           |           |           | data element identifier octet (Vendor defined octet)

```

The attachment structure is as follows:

```

2 octets - attachment type
2 octets - time
2 octets - date
4 octets - size
4 octets - filename prefix (long hex integer)
n octets - ASCII file name (n = Length - 14)

```

4.4.2. CC opcode

```

4C Length 06 02 Length1 name1 02 Length2 name2 etc.
|           |           |           |           |
|           |           |           |           | data (CC data structure)
|           |           |           |           | length of data
|           |           |           |           | Data Element (ASCII string)
|           |           |           |           | field (CC)
|           |           |           |           | length of the following bytes
|           |           |           |           | data element identifier octet (Field octet)

```

The CC data structure is as follows:

```

1 octet - address type
n octets - address (n = Length - 1)

```


4.4.3. COPIES opcode

7F	05	30	20	02	00	00	
							data (short integer value)
							length of data
							Data Element (integer)
							field (COPIES)
							length of the following bytes
							data element identifier octet (Vendor defined octet)

4.4.4. DATE opcode

4C	12	02	28	0F	02	0D	yyyymmdd-hhmm
							data (text line)
							length of string
							Data Element (ASCII string)
							length of data
							Data Element (date)
							field (DATE)
							length of the following bytes
							data element identifier octet (Field octet)

4.4.5. FOLDATTACH opcode

FOLDATTACH is a FIPS field that is used only inside folders, not transmitted messages. It specifies the attachments actually stored with the mail item in the folder as opposed to those sent with it.

7F	Length	35	02	27	FileInfo	02	Length	FileContents
								data (attachment structure)
								length of data
								Data Element (ASCII string)
								field (ATTACHMENTS)
								length of the following bytes
								data element identifier octet (Vendor defined octet)

The attachment structure is as follows:

- 2 octets - attachment type(0=Normal, 1=Document, 2=MacBinary)
- 2 octets - time
- 2 octets - date
- 4 octets - size
- 4 octets - filename prefix (long hex integer)
- 13 octets - ASCIIZ file name

4.4.6. FROM Opcode

The FROM field identifies the originator of the message.

```

4C Length 01 02 Length t h e t e x t
|          |          |          |          |
|          |          |          |          | data
|          |          |          |          | length of data
|          |          |          |          | Data Element (ASCII string)
|          |          |          |          | field (FROM)
|          |          |          |          | length of the data element identifier octet
|          |          |          |          | data element identifier octet (Field octet)

```

4.4.7. HOPCOUNT opcode

```

7F 05 33 20 02 00 00
|   |   |   |   |   |   |
|   |   |   |   |   |   | data (short integer value)
|   |   |   |   |   |   | length of data
|   |   |   |   |   |   | Data Element (integer)
|   |   |   |   |   |   | field (HOPCOUNT)
|   |   |   |   |   |   | length of the following bytes
|   |   |   |   |   |   | data element identifier octet (Vendor defined octet)

```

4.4.8. HOPTRACE opcode

```

7F Length 34 02 Length t h e h o p t r a c e
|          |   |   |   |   |   |   |   |   |
|          |   |   |   |   |   |   |   |   | data
|          |   |   |   |   |   |   |   |   | length of data
|          |   |   |   |   |   |   |   |   | Data Element (ASCII string)
|          |   |   |   |   |   |   |   |   | field (HOPTRACE)
|          |   |   |   |   |   |   |   |   | length of the following bytes
|          |   |   |   |   |   |   |   |   | data element identifier octet (Vendor defined octet)

```

MESSAGE FORMAT opcode

```

7F 05 32 20 02 01 00
|   |   |   |   |   |   |
|   |   |   |   |   |   | data (always 01 00)
|   |   |   |   |   |   | length of data
|   |   |   |   |   |   | Data Element (integer)
|   |   |   |   |   |   | field (MESSAGE FORMAT)
|   |   |   |   |   |   | length of the following bytes
|   |   |   |   |   |   | data element identifier octet (Vendor defined octet)

```

4.4.9. NLSTAG

```

7F 0B 38 02 08 52 03 02 00 00 00 00
|   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   | data (see structure below)
|   |   |   |   |   |   |   |   |   |   | length of data
|   |   |   |   |   |   |   |   |   |   | Data Element (ASCII string)

```

```

|      |      field (NLSTAG)
|      |      length of the following bytes
data element identifier octet (Vender defined octet)

```

The purpose of NLSTAG is to identify the language, the active code page, and other language-related information that is attributed to a message. It contains a Language Code which is defined in NLS.H. Since the concept of "Active Code Page" is IBM-biased, an arbitrary Code Page Number was defined for the Standard Macintosh Character Set (as in IBM PCs, there are many sets for the Mac).

```

// National language support structure  NLS_TAG
#define NLS_FUTURE  5
typedef struct nlstype
{
    unsigned int  codepage; /* active codepage number */
    unsigned char language; /* language code number (defined in NLS.H)
*/
    char future[NLS_FUTURE]; /* reserve */
}
    NLS, *PNLS;
// National Language Support Definitions  NLS_TAG
// Language Code Numbers
#define NLS_UND  0  /* Undefined          */
#define NLS_ENG  1  /* General English      */
#define NLS_USA  2  /* US English          */
#define NLS_FRA  3  /* General French       */
#define NLS_FRC  4  /* Canadian French     */
#define NLS_ESP  5  /* Spanish              */
#define NLS_ITA  6  /* Italian              */
#define NLS_NED  7  /* Dutch                */
#define NLS_DEU  8  /* German               */
#define NLS_SVE  9  /* Swedish              */
#define NLS_POR  10 /* Portuguese           */
#define NLS_KOR  11 /* Korean               */
#define NLS_JAP  12 /* Japanese             */
#define NLS_CHI  13 /* Traditional Chinese  */
#define NLS_CHS  14 /* Simplified Chinese   */

// Code Page Numbers
#define CP_IBM_LATIN1  850 /* IBM ASCII Latin-1 Code Page */
#define CP_MAC_ROMAN  10850 /* Macintosh Standard Roman Character Set
*/

```

4.4.10. PRIORITY opcode

4C	04	18	02	01	Priority
					data (message priority)
				length of data	
			Data Element (ASCII string)		
		field (PRIORITY or precedence)			
	length of the following bytes				
data element identifier octet (Field octet)					

4.4.11. SUBJECT opcode

4C	Length	07	02	Length	t h e	t e x t
						data
				length of data		
			Data Element (ASCII string)			
		field (SUBJECT)				
	length of the following bytes					
data element identifier octet (Field octet)						

4.4.12. TEXT opcode

4C	Length	04	02	Length1	line1	02	Length2	line2	etc.
									data (text line)
					length of data				
			Data Element (ASCII string)						
		field (TEXT)							
	length of the following bytes								
data element identifier octet (Field octet)									

4.4.13. TEXTATTR opcode

7F	Length	36	02	Length	attribute list
					data (attribute structures)
				length of data	
			Data Element (ASCII string)		
		field (TEXTATTR)			
	length of the following bytes				
data element identifier octet (Vendor defined octet)					

The attribute structure is as follows:

- 1 octets - text mode attribute
- 4 octets - run position

5. Algorithms

The following are some of the algorithms that are used for some of the basic functions of the Network Courier programs. This will help clarify the function of some of the more common database files.

5.1. Logging in

```
ok = false
open ( ACCESS2.GLB )
while ( there is another record in the file )
{
    read the record
    if ( the record is not deleted )
    {
        add the record to the list of users in memory
        if ( the user id is the same as that type at logon )
            ok = true
    }
}
close ( ACCESS2.GLB )
return ( ok )
```

5.2. Reading Mailbag Headers

```
open ( nnnnnnnn.KEY )
read the KEY file into bitmap structure
close ( nnnnnnnn.KEY )

open ( nnnnnnnn.MBG )
while ( there is another record in the file )
{
    read the record
    check bitmap to see if record is deleted
    if ( the record is not deleted )
        add the record to the in-memory list of headers
}
close ( nnnnnnnn.MBG )
```

5.3. Sending Mail To Mailbags

```
if ( mail has attachments )
{
    send any attachments to postoffice
    if ( any errors sending attachments )
    {
        delete any attachments successfully sent
        return ( error )
    }
}
```



```

    }

    resolve any groups into members
    update copies field of message
    // one copy per local user + one per external PO or gateway

    send mail to postoffice
    if ( any errors sending mail file )
    {
        delete any attachments successfully sent
        return ( error )
    }

    for ( each user and external PO and gateway addressed )
        add a new mail header to the mailbag file

```

5.4. Checking For Outgoing Mail (External and gateways)

For the sake of simplicity, gateways are referred to as "postoffices".

```

    read mailbag headers for direct postoffice
    if ( there are any postoffices indirect via this postoffice )
        for ( each indirect postoffice )
            read mailbag headers

    if ( there is any mail to send )
    {
        create P1 file
        send mail and P1 file to destination's INQUEUE.MBG
        update sent log file
    }

```

5.5. Mailer

```

Call mbg_busy to lock inqueue.mbg
queue any mail in inqueue.mbg
deliver mail
delete reference from inqueue.mbg, and decrement copies field of the MAI file
Call mbg_unbusy to unlock inqueue.mbg

```

Appendix A - Extending the Network Courier Database

The Network Courier database can be extended without breaking the existing user agents. Two different methods of extending the database are the definition of new FIPS message fields, and allocation of unused space in the database structures.

Each mail message consists of a sequence of FIPS fields. FIPS fields may be any length, and they may contain either FIPS fields or arbitrary binary data. Text color is an example of a new FIPS field which was added to a message without breaking the existing user agents. New user agents will display messages in color, while old user agents will ignore the text color field and display messages in monochrome.

If you want to define a new FIPS field, mail me a message with a short description of the field contents. I will assign you a FIPS field identifier and add the new FIPS field to the database document. This is easy to do, so you can define as many FIPS fields as desired.

Existing user agents will ignore unknown FIPS fields.

The current version of EXTERNAL will ignore all new FIPS fields. Unfortunately, due to time constraints, the changes to EXTERNAL required to send around arbitrary FIPS fields did not make it into the current release. This feature is planned for the next release.

The current version of the X.400 gateway does transport arbitrary FIPS fields. If two post offices are connected by X.400 gateways, then messages transported between the post offices will be encapsulated.

Meanwhile, you can define new FIPS fields and have them exchanged between users on a single post office. If you connect your post offices using X.400 gateways instead of EXTERNAL, you can exchange new FIPS fields between users on different post offices.

When writing to the database, the unused space in database records is always set to zero. When reading the database, the contents of the unused space is ignored. New user agents can store data in the unused space, as long as they accommodate the behaviour of older user agents. If a new field is zero, the new user agents should have appropriate default behaviour.

Unused space in database records is a scarce resource. Look through the database document to find the best place to store your data. Send me a message describing the location of the bytes you wish to allocate, a description of the intended use, and a really good reason for storing the data in an existing structure instead of in a separate file. I will submit your request to the appropriate people for approval.